# Name: IPromise

**Category: Reverse Engineering**

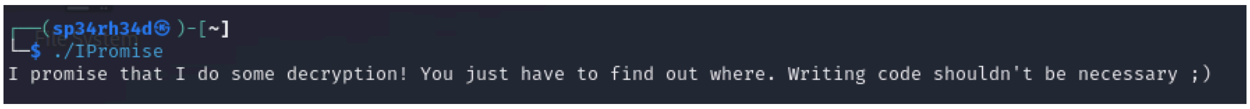
**Difficulty: easy**

**Description: Instead of making the next IPhone, I made this challenge. I do make a truthful promise though...**

## Procedure

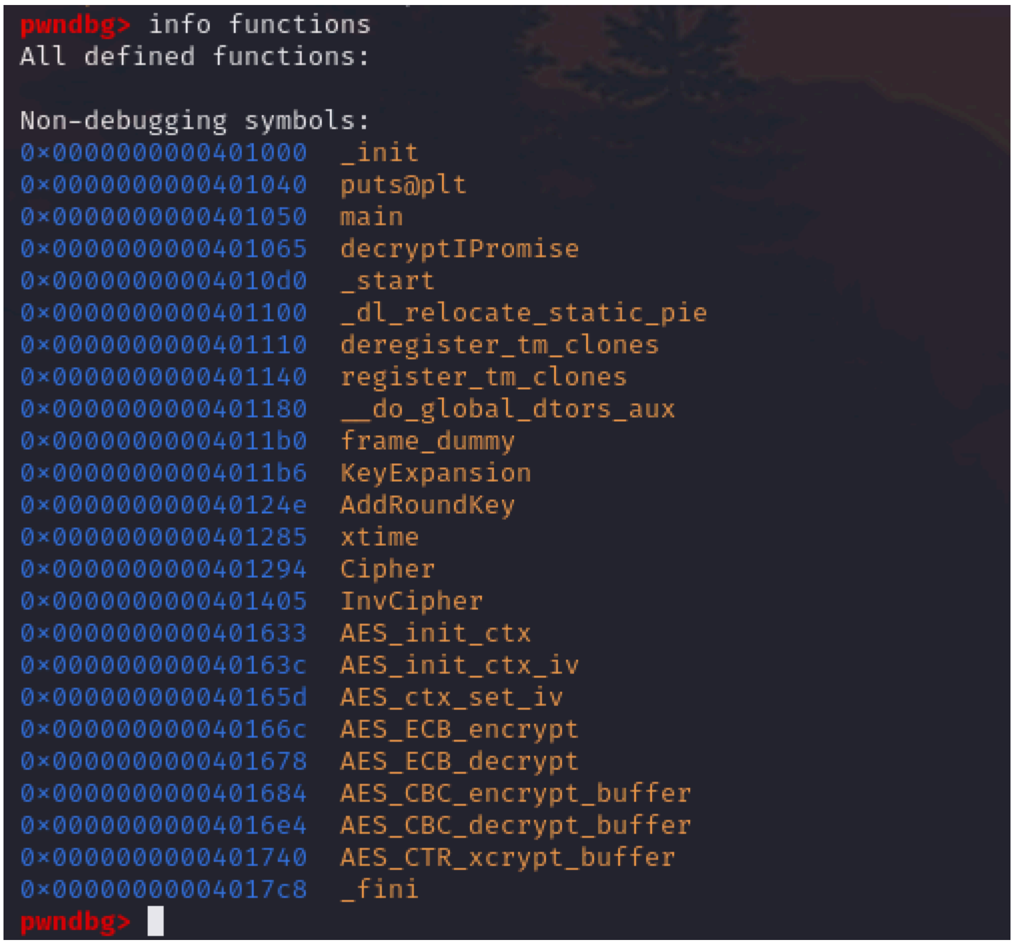
Run `file IPromise` command
Output `IPromise: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=89878e2c4353d02a9ae4a40d8c831124197d2e30, for GNU/Linux 3.2.0, not stripped`

I have ran the binary file to check the behavior, and i see the following output



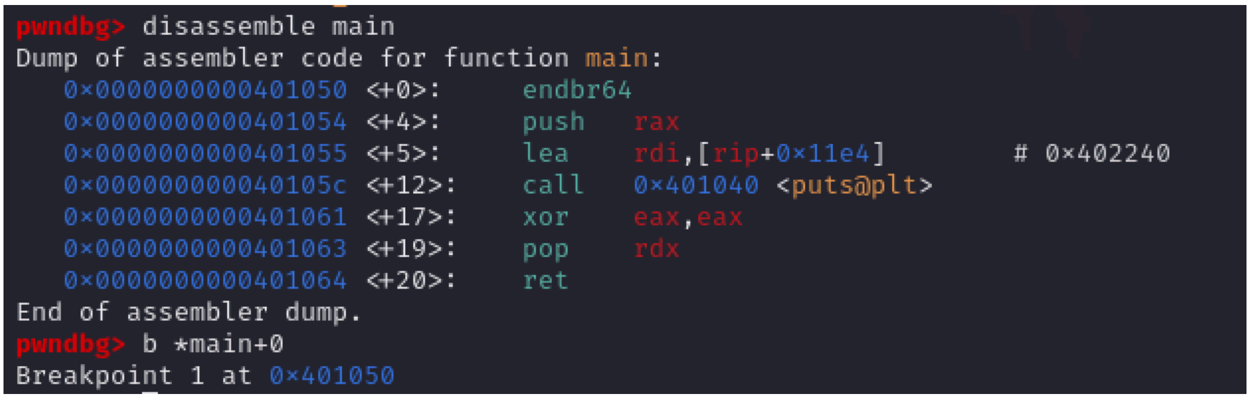
```
┌──(sp34rh34d㉿ )-[~]
└─$ ./IPromise
I promise that I do some decryption! You just have to find out where. Writing code shouldn't be necessary ;)
```

Open the bin file with `gdb ./IPromise`, and list the functions with command `info functions`, We can see `decryptIPromise` function.



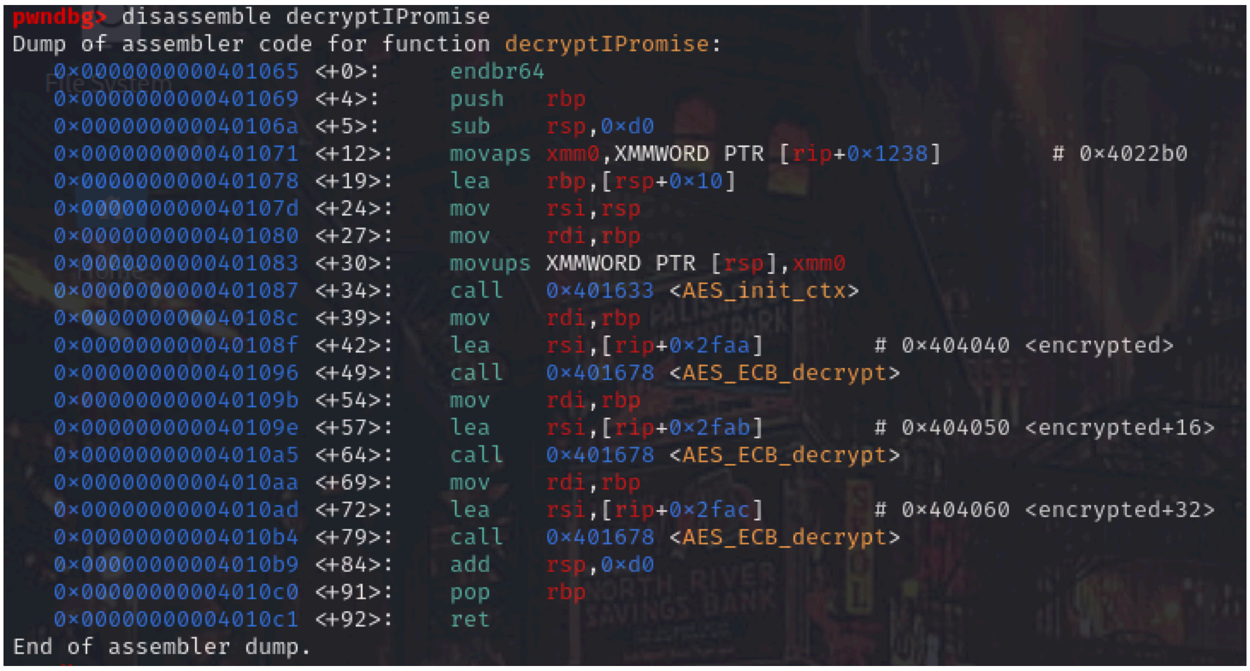
```
pwndbg> info functions
All defined functions:

Non-debugging symbols:
0x0000000000401000  _init
0x0000000000401040  puts@plt
0x0000000000401050  main
0x0000000000401065  decryptIPromise
0x00000000004010d0  _start
0x0000000000401100  _dl_relocate_static_pie
0x0000000000401110  deregister_tm_clones
0x0000000000401140  register_tm_clones
0x0000000000401180  __do_global_dtors_aux
0x00000000004011b0  frame_dummy
0x00000000004011b6  KeyExpansion
0x000000000040124e  AddRoundKey
0x0000000000401285  xtime
0x0000000000401294  Cipher
0x0000000000401405  InvCipher
0x0000000000401633  AES_init_ctx
0x000000000040163c  AES_init_ctx_iv
0x000000000040165d  AES_ctx_set_iv
0x000000000040166c  AES_ECB_encrypt
0x0000000000401678  AES_ECB_decrypt
0x0000000000401684  AES_CBC_encrypt_buffer
0x00000000004016e4  AES_CBC_decrypt_buffer
0x0000000000401740  AES_CTR_xcrypt_buffer
0x00000000004017c8  _fini
pwndbg> 
```

now run `disassemble main` command to check the main function code, then add a breakpoint with `b *main+0` command



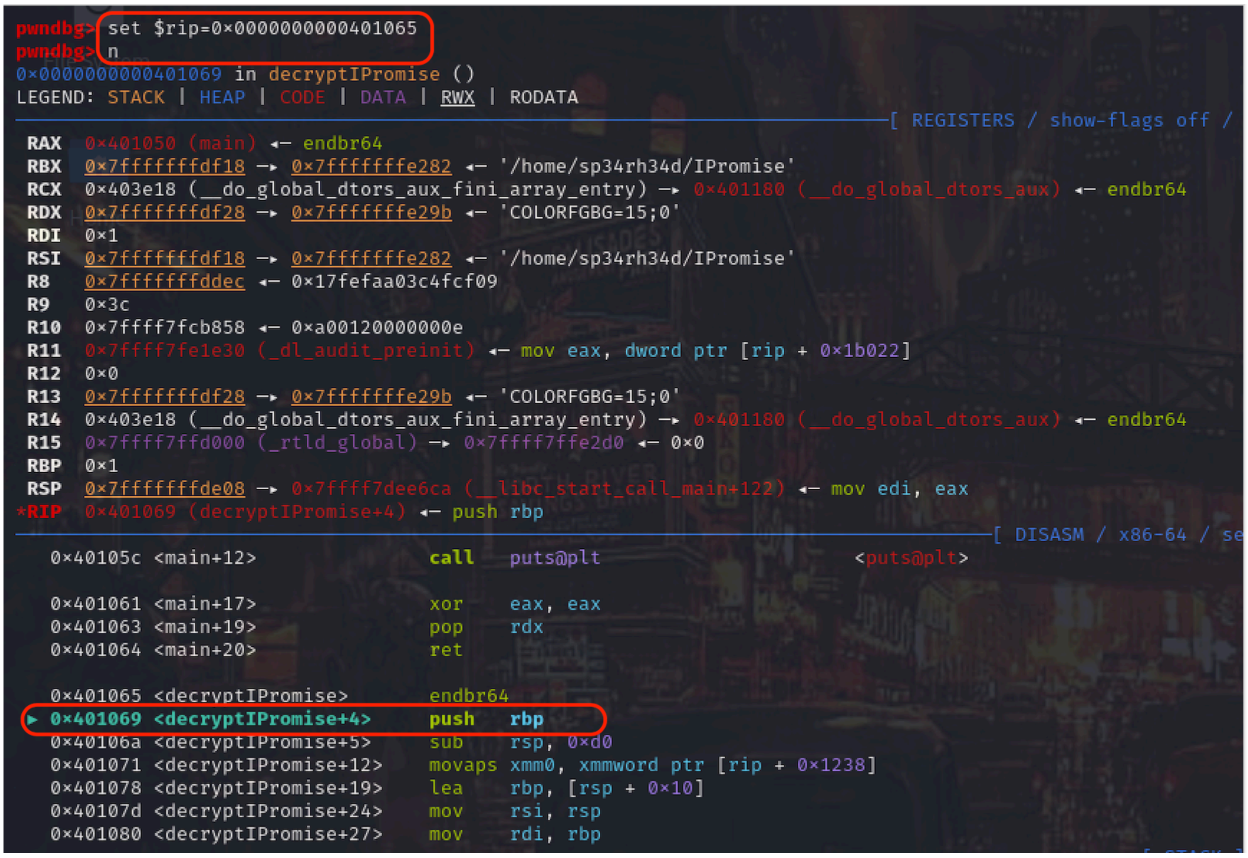
```
pwndbg> disassemble main
Dump of assembler code for function main:
   0x0000000000401050 <+0>:     endbr64
   0x0000000000401054 <+4>:     push   rax
   0x0000000000401055 <+5>:     lea    rdi,[rip+0x11e4]        # 0x402240
   0x000000000040105c <+12>:    call   0x401040 <puts@plt>
   0x0000000000401061 <+17>:    xor    eax,eax
   0x0000000000401063 <+19>:    pop    rdx
   0x0000000000401064 <+20>:    ret
End of assembler dump.
pwndbg> b *main+0
Breakpoint 1 at 0x401050
```

then run `disassemble decryptIPromise` , here we want to know what is the decryptIPromise address, and we can see `0x0000000000401065`



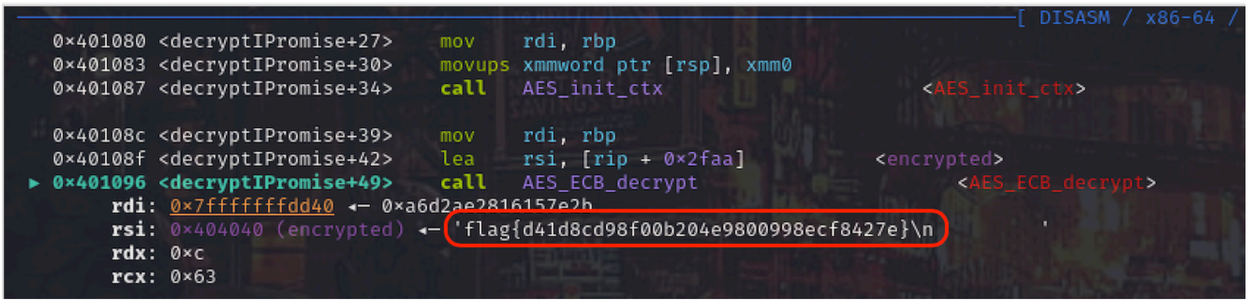
```
pwndbg> disassemble decryptIPromise
Dump of assembler code for function decryptIPromise:
   0x0000000000401065 <+0>:     endbr64
   0x0000000000401069 <+4>:     push   rbp
   0x000000000040106a <+5>:     sub    rsp,0xd0
   0x0000000000401071 <+12>:    movaps xmm0,XMMWORD PTR [rip+0x1238]        # 0x4022b0
   0x0000000000401078 <+19>:    lea    rbp,[rsp+0x10]
   0x000000000040107d <+24>:    mov    rsi,rsp
   0x0000000000401080 <+27>:    mov    rdi,rbp
   0x0000000000401083 <+30>:    movups XMMWORD PTR [rsp],xmm0
   0x0000000000401087 <+34>:    call   0x401633 <AES_init_ctx>
   0x000000000040108c <+39>:    mov    rdi,rbp
   0x000000000040108f <+42>:    lea    rsi,[rip+0x2faa]        # 0x404040 <encrypted>
   0x0000000000401096 <+49>:    call   0x401678 <AES_ECB_decrypt>
   0x000000000040109b <+54>:    mov    rdi,rbp
   0x000000000040109e <+57>:    lea    rsi,[rip+0x2fab]        # 0x404050 <encrypted+16>
   0x00000000004010a5 <+64>:    call   0x401678 <AES_ECB_decrypt>
   0x00000000004010aa <+69>:    mov    rdi,rbp
   0x00000000004010ad <+72>:    lea    rsi,[rip+0x2fac]        # 0x404060 <encrypted+32>
   0x00000000004010b4 <+79>:    call   0x401678 <AES_ECB_decrypt>
   0x00000000004010b9 <+84>:    add    rsp,0xd0
   0x00000000004010c0 <+91>:    pop    rbp
   0x00000000004010c1 <+92>:    ret
End of assembler dump.
```

now we can try to jump to `decryptIPromise` function, run the bin file with `r` command, and gdb will take a break on `main+0` , from here we can try to jump to `decryptIPromise` function, just editing the `rip` (instruction pointer) value to `0x0000000000401065` then press `n` command.



```
pwndbg> set $rip=0x0000000000401065
pwndbg> n
0x0000000000401069 in decryptIPromise ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
─────────────────────────────────────────────────────────────────────────[ REGISTERS / show-flags off /
 RAX  0x401050 (main) ◂— endbr64
 RBX  0x7fffffffdf18 —▸ 0x7fffffffe282 ◂— '/home/sp34rh34d/IPromise'
 RCX  0x403e18 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401180 (__do_global_dtors_aux) ◂— endbr64
 RDX  0x7fffffffdf28 —▸ 0x7fffffffe29b ◂— 'COLORFGBG=15;0'
 RDI  0x1
 RSI  0x7fffffffdf18 —▸ 0x7fffffffe282 ◂— '/home/sp34rh34d/IPromise'
 R8   0x7fffffffddec ◂— 0x17fefaa03c4fcf09
 R9   0x3c
 R10  0x7ffff7fcb858 ◂— 0xa00120000000e
 R11  0x7ffff7fe1e30 (_dl_audit_preinit) ◂— mov eax, dword ptr [rip + 0x1b022]
 R12  0x0
 R13  0x7fffffffdf28 —▸ 0x7fffffffe29b ◂— 'COLORFGBG=15;0'
 R14  0x403e18 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401180 (__do_global_dtors_aux) ◂— endbr64
 R15  0x7ffff7ffd000 (_rtld_global) —▸ 0x7ffff7ffe2d0 ◂— 0x0
 RBP  0x1
 RSP  0x7fffffffde08 —▸ 0x7ffff7dee6ca (__libc_start_call_main+122) ◂— mov edi, eax
*RIP  0x401069 (decryptIPromise+4) ◂— push rbp
──────────────────────────────────────────────────────────────────────────────────[ DISASM / x86-64 / se
   0x40105c <main+12>                call   puts@plt                    <puts@plt>

   0x401061 <main+17>                xor    eax, eax
   0x401063 <main+19>                pop    rdx
   0x401064 <main+20>                ret

   0x401065 <decryptIPromise>        endbr64
 ► 0x401069 <decryptIPromise+4>      push   rbp
   0x40106a <decryptIPromise+5>      sub    rsp, 0xd0
   0x401071 <decryptIPromise+12>     movaps xmm0, xmmword ptr [rip + 0x1238]
   0x401078 <decryptIPromise+19>     lea    rbp, [rsp + 0x10]
   0x40107d <decryptIPromise+24>     mov    rsi, rsp
   0x401080 <decryptIPromise+27>     mov    rdi, rbp
```

press enter until you see the flag in rsi value.



```
────────────────────────────────────────────────────────────────────[ DISASM / x86-64 /
   0×401080 <decryptIPromise+27>    mov    rdi, rbp
   0×401083 <decryptIPromise+30>    movups xmmword ptr [rsp], xmm0
   0×401087 <decryptIPromise+34>    call   AES_init_ctx                    <AES_init_ctx>

   0×40108c <decryptIPromise+39>    mov    rdi, rbp
   0×40108f <decryptIPromise+42>    lea    rsi, [rip + 0×2faa]         <encrypted>
 ▶ 0×401096 <decryptIPromise+49>    call   AES_ECB_decrypt                    <AES_ECB_decrypt>
        rdi: 0×7fffffffdd40 ◂— 0×a6d2ae2816157e2b
        rsi: 0×404040 (encrypted) ◂— 'flag{d41d8cd98f00b204e9800998ecf8427e}\n
        rdx: 0×c
        rcx: 0×63
```

flag `flag{d41d8cd98f00b204e9800998ecf8427e}`